PSCG

Premium Software Consulting Group

(De)Bootstrapping Systems

Considerations Building and optimizing Ubuntu and Debian based root file systems

EMBEDDED OPEN SOURCE SUMMIT

Seattle

April 17, 2024

Ron Munitz

The PSCG

About://Ron Munitz

- Lead: PSCG Holdings → We build companies and invest in stuff.
- Lead: The PSCG → We build products and train engineers to be amazing
- Expertise: Embedded, OS Internals, Security, Startups, "Everything Linux"
- Now doing: Android Internals and Embedded Linux building and training
  - Lately mostly building using : AOSP, Yocto Project, and Debootstrap(!)
- First time at ELC/ABS: 2013, San Francisco
- Last public talk: Embedded Israel, 2024
  - I have known the organizer for a while
  - // sed 's:/.../EOSS 2024!/'

PSCG

Happy to work with you if you are

- A good person
- Honest and direct
- Say what you think and do what you say.

Personal email: ron@thepscg.com

Why am I giving this talk?

That's a good question. Let's prepare a slide about it.

PSCG

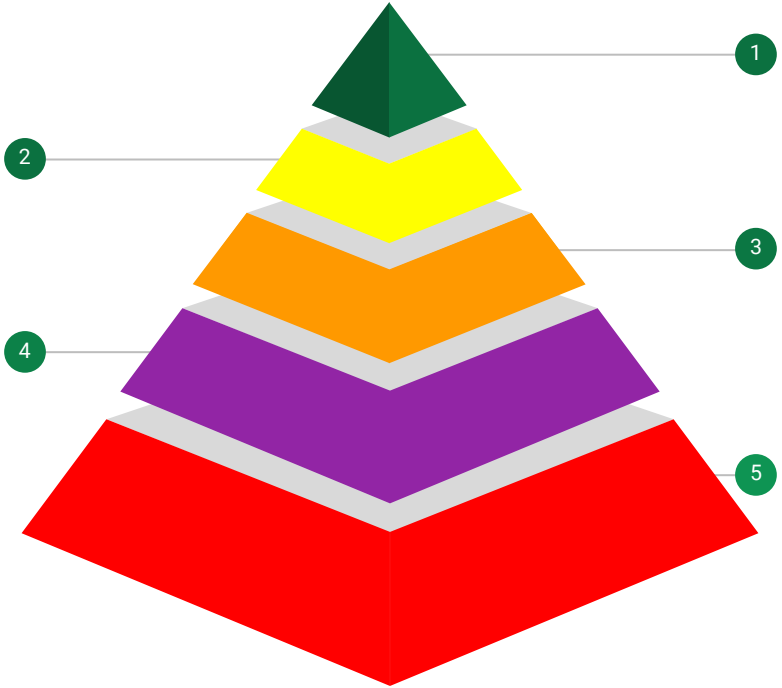
Embedded Linux Learning Objectives

Provide the mechanisms of running binaries

Have a file system and set of tools that enable the user to run that application, including the components (e.g. libraries) that support it

Bootloaders to boot the kernel and initial file system

Bridge between the hardware factory boot code, and the prospective Operating System



Build and Run Applications/Binaries

An executable you can run, and (almost always) has dependencies.

GPOS (Linux) Kernel and Initial root filesystem

Provide the mechanisms to interact with the hardware, provide services to the mechanisms above, and implement multi-process and multi-user mechanisms

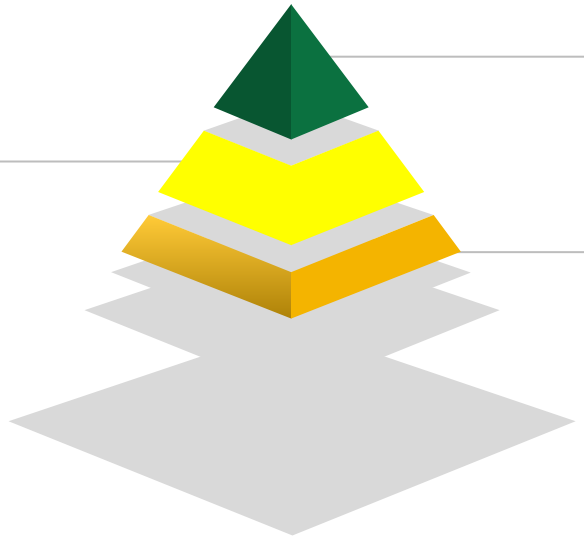
Hardware

This is what you can hold in your hands, or otherwise what "physically exists"

Today we mostly focus on

Provide the mechanisms of running binaries

Have a file system and set of tools that enable the user to run that application, including the components (e.g. libraries) that support it



Build and Run Applications/Binaries

An executable you can run, and (almost always) has dependencies.

GPOS (Linux) Kernel and Initial root filesystem

Provide the mechanisms to interact with the hardware, provide services to the mechanisms above, and implement multi-process and multi-user mechanisms

Objectives

- Meet Debootstrap (and cross-debootstrap)
- Exploring different (Debian based) distros
- Understanding the different package repositories and cached files
- Installing more packages
- Common challenges and solutions in debootstrapping
- Common recipes: default password, motd/issue, ssh, autologin
- Considerations in booting a debootstrapped system
- Sizing and performance constraints: Build-time and runtime (and recovery)
- Runtime tricks: setting up (ethernet) networking in ramdisk and switch_rooting
- Extra: display and audio forwarding

Prerequisites: What you are expected to know

- The attendees have working knowledge of basic Linux command line
- The attendees have a working knowledge of basic **apt** commands such as:
  - *apt-get update*
  - *apt-get install*
  - *apt-get remove*
- Nice to have:
  - Basic knowledge of *dpkg*
    - if you don't hold such, take the time to run **$ sudo dpkg -l** and quickly observe some of the output)
  - Experience with *chroot* or with Linux containers
- If you are used to non *Debian* distros, but feel comfortable with package management, you will work out the differences trivially. Don't worry, you fit in.

Disclaimers: Practical, not necessarily accurate

- Debootstrap has quite a few options, and *cross-debootstrapping* requires additional setup phases (e.g. setting of *qemu-user-static*, registering binfmts etc.)
- We show on an *x86_64* host, but it works in an *arm64* host as well
- It is possible, that some architectures do not support a one-shot debootstrap
  - Then you will use the two step version: **--foreign** and **--second-stage**
- With the exception of RISC-V examples (although it was declared to be officially supported, it still isn't as per early 2024), we will not use that.
  - It is actually supported now, but not in the main debian FTP, but rather in others
  - It is on the other hand trivially supported in the *Ubuntu* distro via *ubuntu-ports* exactly as *arm64* is.

Overall Objective

Create a Debian based root file system, following the Debian packaging conventions and tools, with minimal effort

Meet Debootstrap

Creating a Debian/Ubuntu based root filesystem

Creating Debian bases systems

- Option #1: Get a ready distro. Then start working it out and repackage it.
  - You would be **amazed** on how many people do it
  - They have some "good" reasons. The real reason - incompetence
  - Another leading reason: NVIDIA (and yet, incompetence)
  - Sometimes the good reasons are good enough. But objectively, not very professional.
- Option #2: **debootstrap**
- Option #3: multistrap
  - Why the talk is not about it ⇒ in a later slide
- Other options exist, but we won't get into them.
  - For example, the Yocto Project can use the debian package manager. But it does not build a Debian system (unless you are really eager to over customize it to achieve the wrong goal)

Debootstrap

- A tool which creates *Debian* root filesystems
- Dependency installation in *Debian* based host system:
  $ sudo apt-get install -y debootstrap
- Usage:
  **$ sudo debootstrap **
  **--arch=${deb_arch} **
  **--variant=${deb_variant} **
  **${deb_version} ${target_rootfs_dir} **
  ${deb_url}
- Man pages:
  $ man 8 debootstrap

Debootstrap outcome

The following is a top level view of a *mantic riscv64 minbase* debootstrap output

```
total 60
lrwxrwxrwx  1 root root    7 Mar 18 16:40 bin -> usr/bin
drwxr-xr-x  2 root root 4096 Oct 10  2023 boot
drwxr-xr-x  4 root root 4096 Mar 18 16:40 dev
drwxr-xr-x 30 root root 4096 Mar 18 16:42 etc
drwxr-xr-x  2 root root 4096 Oct 10  2023 home
lrwxrwxrwx  1 root root    7 Mar 18 16:40 lib -> usr/lib
drwxr-xr-x  2 root root 4096 Mar 18 16:40 media
drwxr-xr-x  2 root root 4096 Mar 18 16:40 mnt
drwxr-xr-x  2 root root 4096 Mar 18 16:42 opt
drwxr-xr-x  2 root root 4096 Oct 10  2023 proc
drwx------  2 root root 4096 Mar 18 16:40 root
drwxr-xr-x  4 root root 4096 Mar 18 16:40 run
lrwxrwxrwx  1 root root    8 Mar 18 16:40 sbin -> usr/sbin
drwxr-xr-x  2 root root 4096 Mar 18 16:40 srv
drwxr-xr-x  2 root root 4096 Oct 10  2023 sys
drwxrwxrwt  2 root root 4096 Mar 18 16:42 tmp
drwxr-xr-x 11 root root 4096 Mar 18 16:40 usr
drwxr-xr-x 11 root root 4096 Mar 18 16:40 var
```

Multistrap in a nutshell

- Multistrap is decent
  - Multistrap can be seen as an "enhanced debootstrap"
  - It can use several package sources (think of it as apt sources)
  - And can run hooks across some of the stages
- So what are its disadvantages?
  - It doesn't work out of the box, doesn't respect its own documentation, and is overall annoying
  - Can run only on Debian-like systems (which is not a huge problem)
  - Requires learning more configuration tools
  - Can get too complex to use and maintain if you just want to keep things simple
- Multistrap is out of the scope of this talk, and you can learn more in https://wiki.debian.org/Multistrap

What to do with a deboostrapped system?

- Chroot and do things
- Run in a container (which means also isolate) and do things
- Use as a root filesystem for a target device
  - A Linux one
    - Then, you must take care of a proper *init* file or framework
  - Or a multi-OS one (won't discuss it here, but there can be multiple nice tricks when Linux is not the main host)

Example: Debian Releases

@see https://www.debian.org/releases/

Index of releases (January 7th 2024)

- The next release of Debian is codenamed *trixie* — *testing* — no release date has been set
- Debian 12 (*bookworm*) — current *stable* release
- Debian 11 (*bullseye*) — current *oldstable* release
- Debian 10 (*buster*) — current *oldoldstable* release, under LTS support
- Debian 9 (*stretch*) — archived release, under extended LTS support
- Debian 8 (*jessie*) — archived release, under extended LTS support
- Debian 7 (*wheezy*) — obsolete stable release
- Debian 6.0 (*squeeze*) — obsolete stable release
- Debian GNU/Linux 5.0 (*lenny*) — obsolete stable release
- Debian GNU/Linux 4.0 (*etch*) — obsolete stable release
- Debian GNU/Linux 3.1 (*sarge*) — obsolete stable release
- Debian GNU/Linux 3.0 (*woody*) — obsolete stable release
- Debian GNU/Linux 2.2 (*potato*) — obsolete stable release
- Debian GNU/Linux 2.1 (*slink*) — obsolete stable release
- Debian GNU/Linux 2.0 (*hamm*) — obsolete stable release

Example: URL: http://deb.debian.org/debian

redirects to http://ftp.debian.org/debian/ :

Parent Directory		
README	2023-12-10 17:10	1.2K
README.CD-manufacture	2010-06-26 09:52	1.3K
README.html	2023-12-10 17:10	2.8K
README.mirrors.html	2017-03-04 20:08	291
README.mirrors.txt	2017-03-04 20:08	86
dists/	2023-12-10 17:11	
doc/	2024-01-14 07:55	
extrafiles	2024-01-14 08:25	226K
indices/	2024-01-14 08:24	
ls-lR.gz	2024-01-14 08:18	15M
pool/	2022-10-05 17:09	
project/	2008-11-17 23:05	
tools/	2012-10-10 16:29	
zzz-dists/	2023-10-07 11:07	

You can then, e.g. explore http://ftp.debian.org/debian/dists/stable/main/ to see what architectures are supported by the latest stable (as per April 2024, RISC-V is not here, but rather in *debian-ports*)

Some notes for Browsing a Debian FTP site

- *ls-lR.gz* is a gzipped file of the output of running **$ ls -lR** on the ftp tree
  - The compressed file is 17MB (!) and the ls -lR uncompressed output is 148M (!!) as per April 17,2024. Imagine that!
- Releases are specified under the *dists/* (and the *zzz-dists*) folders
- Each Debian version gets its designated folder (e.g. *dists/bookworm*) where
  - *Changelog* shows what has changed (how surprising…)
  - *Release* has the contents of the files and digests (148KB as per the last time checked) which are a list of compressed archives specifying the packages for several components and architectures
  - *Release.gpg* is a signed digest on the *Release* file
  - *InRelease* is a combination of the two
- You will find that some of the files in *dists* are links to some files in *zzz-dists*
  - The reason for this is imposing a certain order of updates

Example: Debootstrapping Debian Trixie

The following examples were recorded on January 14 2024:

- **# debootstrap --variant=minbase trixie 1** http://deb.debian.org/debian
- Relevant sizes immediately after debootstrapping:
  - 215M 1
  - 50M 1/var/lib/apt
  - 33M 1/var/cache/apt
- After **chroot**-ing and **apt-get clean**-ing (which cleans up */var/cache/apt*)
  - 183M 1
  - 50M 1/var/lib/apt
  - 12K 1/var/cache/apt

Note: the same release, 5 months before that check, had 49M in the package list.
You can be sure that sizes at some point are just representing a point in time.

Example: Debootstrapping Debian Trixie

The following examples were recorded on January 14 2024:

- **# debootstrap --variant=minbase trixie 1 http://deb.debian.org/debian**
  - **215M -** immediately after debootstrapping
  - **183M -** after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      50M /var/lib/apt/
- **# debootstrap trixie 2 http://deb.debian.org/debian**
  - **311M -** immediately after debootstrapping
  - **261M -** after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      50M /var/lib/apt/
- **# diff -r 1/etc/apt/ 2/etc/apt/** shows no diff.
  - Which explains why the mirror lists (most of the contents of */var/lib/apt* is identical)

Example: (cross-)Debootstrapping Debian Trixie

The following examples were recorded on January 14 2024:

- **# debootstrap --arch=arm64 --variant=minbase trixie 1-arm64 \ http://deb.debian.org/debian**
  - **234M -** immediately after debootstrapping
  - **204M -** after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which **50M** /var/lib/apt/
- **# debootstrap --arch=arm64 trixie 2-arm64 http://deb.debian.org/debian**
  - **336M -** immediately after debootstrapping
  - **289M -** after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which **50M** /var/lib/apt/
- **# diff -r 1/etc/apt/ 2/etc/apt/** shows no diff.
  - Which explains why the mirror lists (most of the contents of */var/lib/apt* is identical)

Example: (cross-)Debootstrapping Debian Trixie

The following examples were recorded on August 20 2023:

- **# debootstrap --arch=arm64 --variant=minbase trixie 1-arm64 \ http://deb.debian.org/debian**
  - **224M -** immediately after debootstrapping
  - **195M -** after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which **49M** /var/lib/apt/
- **# debootstrap --arch=arm64 trixie 2-arm64 http://deb.debian.org/debian**
  - **305M -** immediately after debootstrapping
  - **256M -** after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which **49M** /var/lib/apt/
- **# diff -r 1/etc/apt/ 2/etc/apt/** shows no diff.
  - Which explains why the mirror lists (most of the contents of */var/lib/apt* is identical)

Take away from the arm64 example - size matters!

You could see that over 5 months there was:

- 10MB of difference in the *minbase variant*
- 30MB of difference in the default variant.

That's a lot!

- If you have limited storage, you really need to be mindful
- If you are building with some scripts someone made and don't understand what is being done - don't be surprised if you cannot create an image due to exceeding a previously defined size!

⇒ You are an embedded guru - you are expected to know what you're doing!

Example continued: Understanding /etc/apt

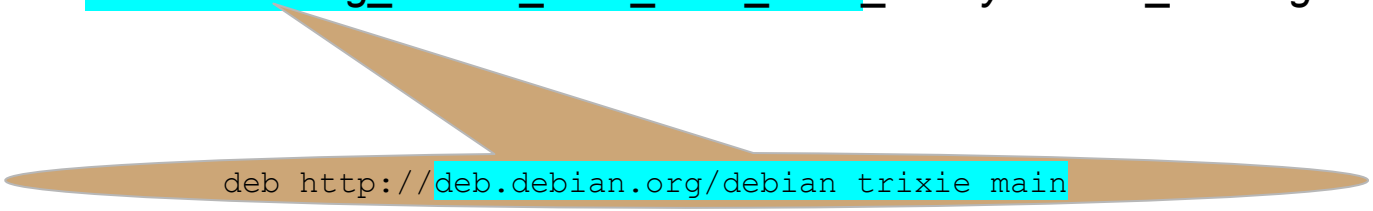
```
$ tree etc/apt/
etc/apt/
├── apt.conf.d
│   ├── 01autoremove
│   └── 70debconf
├── auth.conf.d
├── keyrings
├── preferences.d
├── sources.list
├── sources.list.d
└── trusted.gpg.d
    ├── debian-archive-bookworm-automatic.asc
    ├── debian-archive-bookworm-security-automatic.asc
    ├── debian-archive-bookworm-stable.asc
    ├── debian-archive-bullseye-automatic.asc
    ├── debian-archive-bullseye-security-automatic.asc
    ├── debian-archive-bullseye-stable.asc
    ├── debian-archive-buster-automatic.asc
    ├── debian-archive-buster-security-automatic.asc
    └── debian-archive-buster-stable.asc
```

```
deb http://deb.debian.org/debian trixie main
```

Public keys used for verification (which is why *http* is fine really)

Example: Debootstrapping Debian Trixie (cont.)

- You can see the main file which takes these 49MB (in the previous example) is *var/lib/apt/lists/deb.debian.org_debian_dists_trixie_main_binary-amd64_Packages*



- This is a (**very**) long text file.
  - Observe it
  - And compare its output with the output of **$ apt-cache show** <package name>

A bit about var/cache

Upon initial debootstrap:

- *var/cache/debconf*
- *var/cache/ldconfig* - empty as there is no linker db
  - *ldconfig* uses this to help build */etc/ld.so.cache*
- *var/cache/apt*
  - Has all of the *.deb* packages that were downloaded
  - Has an empty *partial* folder (as there are no partial downloads if it succeeded…)
  - Does not have any lock files (as there are no locks if it succeeded)

Example: Debootstrapping older Debian distros

The following examples were recorded on both Aug 20 2023 and Jan 14 2024:

- **# debootstrap --variant=minbase bookworm 12** http://deb.debian.org/debian
  - **206M -** immediately after debootstrapping
  - **175M -** after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      48M /var/lib/apt/
- **# debootstrap --variant=minbase bullseye 11** http://deb.debian.org/debian
  - **214M -** immediately after debootstrapping
  - **179M -** after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      44M /var/lib/apt/

Example: Debootstrapping Ubuntu Lunar Lobster

The following examples were recorded on August 20 2023:

- **# debootstrap --variant=minbase lunar x86_64-lunar http://archive.ubuntu.com/ubuntu/**
  - **124M** - immediately after debootstrapping
  - **99M** - after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      8.2M /var/lib/apt/
- **# debootstrap --arch riscv64 --variant=minbase lunar riscv64-lunar http://ports.ubuntu.com/ubuntu-ports/**
  - **108M** - immediately after debootstrapping
  - **84M** - after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      6.8M /var/lib/apt/

Example: Debootstrapping Ubuntu Mantic Minotaur

The following examples were recorded on January 14 2024:

- **# debootstrap --variant=minbase mantic x86_64-mantic http://archive.ubuntu.com/ubuntu/**
  - **124M** - immediately after debootstrapping
  - **99M** - after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      7.2M /var/lib/apt/
- **# debootstrap --arch riscv64 --variant=minbase mantic riscv64-mantic http://ports.ubuntu.com/ubuntu-ports/**
  - **113M** - immediately after debootstrapping
  - **88M** - after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      6.9M /var/lib/apt/

Example: Debootstrapping Mantic for arm64 vs arm

- **# debootstrap --arch arm64 --variant=minbase mantic mantic-arm64 http://ports.ubuntu.com/ubuntu-ports/**
  - **144M** - immediately after debootstrapping
  - **120M** - after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      7.1M /var/lib/apt/
- **# debootstrap --arch armhf --variant=minbase mantic mantic-armhf http://ports.ubuntu.com/ubuntu-ports/**
  - **109M** - immediately after debootstrapping
  - **86M**- after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      6.9M /var/lib/apt/

Example: Debootstrapping Lunar for arm64 vs arm

- **# debootstrap --arch arm64 --variant=minbase lunar arm64-lunar http://ports.ubuntu.com/ubuntu-ports/**
  - **143M** - immediately after debootstrapping
  - **119M** - after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      7.1M /var/lib/apt/
- **# debootstrap --arch armhf --variant=minbase lunar arm-lunar http://ports.ubuntu.com/ubuntu-ports/**
  - **115M** - immediately after debootstrapping
  - **92M**- after **# apt-get clean** which cleans up */var/cache/apt*
    - out of which
      7.1M /var/lib/apt/

What can we learn from the previous slides?

- Ubuntu's *main* component holds far less packages than the Debian's one.
- Different architectures have different binary sizes
- Meaning, that if you don't really need 64bit support on your devices, you might as well save some precious storage and memory space by opting in for a 32bit architecture (even if your hardware supports 64 bit)
- Note the different URL's, for the *x86_64* and the other architecture ports
  - Exercise: browse http://archive.ubuntu.com/ubuntu/ and http://ports.ubuntu.com/ to better understand what you can build, port and debootstrap, Ubuntu-wise

More things to be aware of - root permissions

- As you could see, **debootstrap** requires root permissions for obvious reasons
  - If it's not obvious to you: can you **$ chown root:root <somefile>** ?
- There are multiple solutions for that.
  - ["modern"] - use a Debian based container (where someone else already debootstrapped a rootfs), and run debootstrap in it
  - Use fakeroot (a general concept, regardless of debootstrap, super useful in building packages and Linux distros in a non-privileged environment
    - A moment of honesty: no one really does that, but distro builders such as the Yocto Project will do that for you.
- Use debootstrap with *--variant=fakechroot*
  - An example is presented in the next slide, and troubleshooting *fakeroot* and *fakechroot* is left as an exercise. You are unlikely to need to deal with those yourself in your embedded Linux development life (it is complex, and others have solved it for you in the respective build systems, and/or in modern development workflows.)

Example: Using debootstrap with fakechroot

- First attempt:
  $ sudo debootstrap --arch=arm64 --variant=fakechroot trixie 3.1 http://deb.debian.org/debian
  W: Cannot check Release signature; keyring file not available /usr/share/keyrings/debian-archive-keyring.gpg
  E: This variant requires fakechroot environment to be started
- Resolve by installing *fakechroot*
  $ sudo apt-get install -y fakechroot
- Now run *fakeroot* inside a *fakechroot* and use the *fakechroot* debootstrap variant
  $ fakechroot fakeroot debootstrap --arch=arm64 --variant=fakechroot trixie 3.1 http://deb.debian.org/debian
- This will likely fail on /bin/true . Homework: troubleshoot ;-)
  - As this is out of the scope of this particular lecture

OK, so debootstrap succeeded. Great. Now what?

- *chroot*, and knock yourself out
  - [populate the *apt* resource list with your choice and *apt-get update* if you do]
  - Install packages
  - Change configuration files
- Edit directly (out of chroot) files or populate them
- Keep the debootstrapped directory aside for reproducible builds
  - What if somebody changes something in the Debian repositories?

More things to be aware of - security

- While *dpkg* does not verify the source of (any) package, and only breaks upon either internal state or package dependency problems, *apt* does care about security
- It is the responsibility of the maintainer to give *apt* the means of verifying root of trust, via introducing trusted *gpg* keys, and trusted certificates (the *ca-certificates* package is useful)
- Sometimes, for development purposes, it is simpler to avoid security checks
  - Can be useful if you use third party repositories and don't want *apt* to fail if you did not introduce keys yet
  - Can be useful to avoid all kind of annoying debootstrap issues
    - Usually resolved by mounting */tmp, /dev/, /dev/pts* and creating */dev/zero* / */dev/null*
    - Don't ask why, but you'll see it when you'll get there. Hopefully, you won't ever see it!
- Do not do it for production systems if you care about security!

Some common chroot troubleshooting

```
# apt-get update
Get:1 http://ports.ubuntu.com/ubuntu-ports lunar InRelease [267 kB]
0% [Working]/usr/bin/apt-key: 95: cannot create /dev/null: Permission denied
/usr/bin/apt-key: 95: cannot create /dev/null: Permission denied
/usr/bin/apt-key: 95: cannot create /dev/null: Permission denied
E: gpgv, gpgv2 or gpgv1 required for verification, but neither seems installed
Err:1 http://ports.ubuntu.com/ubuntu-ports lunar InRelease
  gpgv, gpgv2 or gpgv1 required for verification, but neither seems installed
Reading package lists... Done
W: GPG error: http://ports.ubuntu.com/ubuntu-ports lunar InRelease: gpgv, gpgv2 or gpgv1
required for verification, but neither seems installed
E: The repository 'http://ports.ubuntu.com/ubuntu-ports lunar InRelease' is not signed.
N: Updating from such a repository can't be done securely, and is therefore disabled by
default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
```

Some solutions: properly mount */dev /dev/pts /tmp*. Give all users rw permissions to */dev/null* and */dev/zero*, Install *ca-certificates*, import gpg keys, …

Some common chroot troubleshooting

- There are obviously things that **you cannot** do with a simple *debootstrap*/*chroot* scenario, even if you can run foreign architectures (i.e. *qemu-user / binfmt_misc* et. al)
- *chroot* itself has no resource isolation cf. containers (namespaces) and of course using a VM (another kernel)
- So everything that is network related, or hardware related, depends on your host
- So the rule of thumb is to set whatever you can **before running on a real**, or a more isolated **target**, and if you are not sure what to do with some configurations - "learn on the target" - and then port it back!

Interim Summary

The important concepts we covered in this part are:

- What debootstrap is
- Differences between different Debian and Ubuntu flavors
- Where debian packages are stored on the servers, and on your rootfs
- How the FTP structure where packages are stored is set
- In Linux, size does matter
  - Now go tell that to everyone porting entire Python and NodeJS code bases to an Embedded Linux device
  - Or to those who maintain, flash, upload, copy, images…

Preparing for target usage

Making your deboostrapped root filesystem work in the real world

Using your debootsrapped rootfs on your device

- Assuming you intend to use it as your rootfs
  - Either directly (root=/…)
  - Or after an **exec switch_root** from a ramdisk (which may have taken care of some things)
- And assuming you want to have a fully fledged *Linux* distro
- You must provide either an *init* script/executable (e.g. as the ramdisk would), or preferably, install an *init framework* such as
  - **systemd** (de facto standard, powerful, heavy)
  - **openrc** (Gentoo's system, harder to install on some *Debian* version)
  - **busybox** (easy, Sys-V like)
  - **finit-sysv** (easy, sysv like, trivial)
  - We will not discuss *upstart* because no one uses it (unless you are eager to run Ubuntu 14.04 or something from that era)
- This opens to a whole set of tradeoffs: speed, image size, flexibility and more.

Using your debootsrapped rootfs on your device

- You may or may not also select to use a logging service such as *rsyslog* or *syslog-ng*
  - This opens to a whole set of tradeoffs: speed, image size, flexibility and more.
- You may or may not want to have your filesystem read only and then
  - Use another partition for writable data
  - Use another partition for writable data with overlays (e.g. use *overlayfs*)
- You may or may not chose to apply additional security measures
  - Of course DAC, have a non-root user name etc.
  - But also MAC, using LSM's such as *selinux*, *apparmor* and the likes, and more
- And you may of course want to select *login credentials*, add *networking* support, decide whether to allow *ssh* access, decide if you want to have *graphics* and *media* support, and more.

Supporting kernel removable modules

- POTUS 2023 and Android pre GKI et. al: "Don't"
  - That's a joke, otherwise there wouldn't be a slide about it
- Either use *busybox* or *kmod* (or implement yourself, there is a system call…)
  - Don't count on the distro busybox though, as *modprobe* is likely not compiled there
- But you must have your system properly setup
  - kernel in-tree modules: make modules_install
  - out-of-tree modules: modules install + **depmod**
    - Build time: cross-depmod
      $ depmod -b $target_dir -F $your_build/System.map $depmod_kv
    - Runtime:
      # depmod

Supporting kernel removable modules - size

- Providing an entire build directory to your "users" works for OOT module buildings
  - sorry about the quotes, but "our" users are not exactly "their" users
- But at what cost?
- What if you only need to provide them with the option to build your code, and never worry* about the consequences?
  - They should worry. But they should trust you
  - (hmm… trust no one.. *sigh* I'll never get out of here if I keep the disclaimers)
- **$ make modules_prepare** for the rescue
  - Ship < 10M folder artifact that builds in a couple of seconds
  - Instead of > 1GB that builds in more
  - Then you can **$ make -C <that folder> M=$(pwd) modules,** cross-depmod, and ship
  - But you need to be specific and use *KBUILD_MODPOST_WARN=1*

Supporting kernel removable modules - size

- **$ make modules_prepare** for the rescue
  - Ship < 10M folder artifact that builds in a couple of seconds
  - Instead of > 1GB that builds in more
- Then you can
  - **$ make -C <that folder> M=$(pwd) modules**
  - install or copy to your target rootfs in whatever means you see fit
  - cross-depmod
  - and ship
  - But you need to be specific and use *KBUILD_MODPOST_WARN=1*
  - Without it you get the following warning followed by a very justified build failure:

```
WARNING: Module.symvers is missing.
         Modules may not have dependencies or modversions.
         You may get many unresolved symbol errors.
         You can set KBUILD_MODPOST_WARN=1 to turn errors into warning
         if you want to proceed at your own risk.
```

Let's add some packages to a specific distro over a practical example

Practical Example: Ubuntu Lunar root filesystem

In the next couple of slides we will:

- Discuss some tradeoffs and give recipes for configuring an *U*buntu *Lunar Lobster (23.04)* root filesystem
- Describing common tasks applicable literally to every distro and setup
- Making it very easy for you to follow up on making a bootable and workable root filesystem

Note that:

- Most of these steps are easily applicable elsewhere
- Bootloaders, Linux kernel, and initial ramdisk are out of the scope of these recipes (but are mentioned for reasoning)

Minimal Networking requirements

- We will **not address** Wireless and Bluetooth at this point. But whatever is said about Ethernet, essentially applies to them as well
  - For those, install the *wpasupplicant* and *bluez* packages respectively
  - And/or decide on a network manager such as *networkmanager* or *netplan* or more
- A **very important** thing to understand as per networking:
  Once userspace configures and/or reconfigures (as in re-exchanging secrets etc.), networking then it is **up to the Kernel** to do the work itself
  - It is so important that we aer going to repeat it in the next slide

Minimal Networking requirements

- A very important thing to understand as per networking:
  Once the userspace configures and/or reconfigures (as in re-exchanging secrets etc.), networking then it is **up to the Kernel** to do the work itself
- Meaning that if you had a minimal ramdisk that used *busybox*'s *udhcpc* to get an *ip address* and set the *default routes*, you will have network after *switch_root/chroot*
- In that case, you don't need anything to support networking in a minimal debian root file system except for setting the name servers in */etc/resolv.conf* if you don't use more complex DNS frameworks

Minimal Networking requirements

Assuming you do want to do some meaningful networking activities in your root file system. You would want to install the following packages (via *apt*):

- **iproute2** - for the *ip* util. This alone allows:
  - static ip setting
  - routing rules
- **isc-dhcp-client** - a DHCP client. Dependent on *iproute2*
- **udhcpc** / **busybox** a small footprint busybox based DHCP client
  - In Ubuntu, it requires the *universe* component, which would add significant cache sizes
    - over 150MB before cleaning caches
    - 80MB freed by apt-get clean for both main and universe
    - 102MB in apt-caches for both main and universe
  - If you care about size - you can build busybox yourself and optimize accordingly
- By using busybox you can get both the *ip* utility and a dhcp client (and more)

Minimal network requirements - observation

- By using *busybox* you can get both the *ip* utility and a dhcp client.
- And more useful networking utils (*ping, traceroute, nc, telnet* to name a few)
- If you get to such a point, you may want to ask yourself, if you are really interested in a *Debian* root file system
- There are many tradeoffs
- Handling them are presumably most of what the *Embedded Wizard* does

busybox networking - easy dhcp-ing recipe

- **# udhcpc eth0 -s /thepscg-dhcp-hook.sh**
- The dhcp hook will then need to implement receiving of the ip address (from the *udhcpd* client) and set up both the ip ("as if it were static"), and take care of the route table setting, e.g:

  # cat /thepscg-dhcp-hook.sh

```
#!/bin/sh
if [ ! $1 = bound ] ; then
     echo "$0 $@ argumetns were called. ip=$ip"
     exit 0
fi
ip addr add $ip/$subnet dev $interface
network=$(echo $ip | cut -d"." -f1-3).0
class=24 # could translate subnet but for now hardcoding
ip route add $network/$class dev $interface
ip route add default via $router
```

Minimal set of useful network packages

- You would probably want some convenient means to troubleshoot your networking setup, and perhaps do some more
- Busybox takes care of quite a few of these requirements
- Otherwise, some useful packages would be
  - **iputils-ping**
  - **traceroute**
  - **tcpdump**
  - **telnet**
  - **openssh-client** - ssh client only
  - **ssh** - ssh client and server. Be mindful of the dependencies, to avoid systemd installation
  - **netcat-openbsd** - netcat. There are two netcats, this is the more powerful one.
  - **nmap** - network scanner (size consuming, so be mindful of that)
  - **net-tools** - provides netstat and some other tools for those who don't like the *iproute2* tools

Basic setup - setting up the root user

- In your *chroot* environment set the *root* user password using **# passwd**
- You can also decide to lock/unlock the user password login:
  - **passwd -l root #** locks root from logging in with password
  - **passwd -u root #** unlocks it
- You can add additional users and groups with the **useradd, groupadd, usermod,** and additional tools, or the **adduser** set of utilities.
- You can (and should) do all of these in a *chroot* on the host, before provisioning
  - In fact, if you know your stuff you can populate the files directly, but it is more error prone, and so less recommended

Remember that someone needs to call *login* though. This would usually be done by a variant of *getty* called by the *init* framework (or by a login script)

Greeting the user upon login (motd, issue)

- Before logging in:
  - */etc/issue* contains a message to be shown before a (local) login
  - */etc/issue.net* contains a message to be shown before a *telnet* or *rlogin* session
    - But it will not show before an *ssh* login unless explicitly configured
    - To show a message before an *ssh* login in *openssh-server* add the following to */etc/ssh/sshd_config* (you can select any file, doesn't have to be */etc/issue.net*)

      ```
      Banner /etc/issue.net
      ```
- After logging in (run in this order)
  - */etc/update-motd.d* contains a list of files **to be executed** in alphanumeric order (@see *run-parts* for implementing something like this thoughtlessly and trivially)
  - */etc/motd* contains a single message of the day

Basic setup -setting up ssh access

- The following is absolutely not recommended, security-wise!
- If you choose to install the **ssh** package, you can configure root ssh login with password follows by adding the following to */etc/ssh/sshd_config*

  `PermitRootLogin yes`
  - Careful - avoid install recommends if you are not interested in ~100MB installation of systemd.
  - Careful - there is an *ssh_config* and an *sshd_config* file. Do not confuse them!
- You may test then on the target itself.
  - First, the *sshd* must be running. You can run it usually by starting service *ssh* (e.g. with *systemctl, service* or another relevant command, or by direct invocation)
  - Then, you can go ahead and try to **# ssh localhost**
  - Once you verified self ssh works, you can troubleshoot networking issues, and eventually ssh into your machine

We are only mentioning the basics of the basics, this is not even scratching the surface of ssh configuration

Caveats: apt and DNS

- When you debootstrap for the first time, you usually do not need to do anything special and accessing the apt mirror just works
- But what if it doesn't?
  - Classical example: you test your system, it works with some DNS server, and then you try to reuse the file system
  - Classical example: You use something provided by someone else, and got their */etc/resolv.conf* populated (either with, or without *systemd-resolved*)
- The good thing is that it becomes very obvious and *apt* lets you know when it cannot access an address
  - It could be a real error, but it could also be inaccessible mirrors
  - It could also be broken installs, bad or no-longer-existing keys, etc…
  - In time and experience you will be able to easily troubleshoot

Caveat: apt, DNS and debootstrap

A classical example:

- You run your rootfs either on the device or with QEMU
- */etc/resolv.conf* gets populated
  - (e.g. with *nameserver 10.0.2.3 in* QEMU - which is obviously not accessible from your host)
- You want to fix something inside *debootstrap* + *chroot*
- You get no network…

Knowing this kind of caveats will have you resolve the issue in 10 seconds.

busybox init - example /etc/inittab

- If you use */sbin/init* as busybox, **without** and */etc/inittab* file it will implement some default rules, using something called *askfirst*
  - It will tell you to enter any key and will open a shell on the same terminal you are in
- Otherwise, you can implement some directives
- For example, this is a legit login-less enabling */etc/inittab*:

```
::sysinit:/bin/echo -e "\x1b[42mThePSCG says: Hello sysinit!\x1b[0m"
::respawn:-/bin/sh
::sysinit:/network/udhcpc-wrapper-script.sh
```

Quick and dirty init using finit

- From the *universe* component install the package *finit-sysv*
- Add a tty line for your console tty, to spawn login on it:
  `tty [12345] /dev/ttyAMA0 linux noclear nowait`
- Or if you want to skip login altogether, just auto-login as root
  `tty [12345] /dev/ttyAMA0 linux noclear nowait nologin`

Every init framework will have its own ways to achieve some of these tasks.
Every device will naturally have its own serial device for console, so *ttyAMA0* may be replaced with another value

Using systemd

- The more complex your system become, and the more resourceful, the more likely you are to move to systemd, by installing the *systemd* package.
- It offers many advantages
  - Parallel execution capabilities
  - Debugging and boot time tracing capabilities, recovery capabilities
  - Very well maintained
  - Inherent container spawning (e.g. *systemd-nspawn*)
  - Integral resource management (cgroups et. al)
  - Internal DNS (systemd-resolved)
  - And much more
- But definitely has its disadvantages
  - Size
  - Speed
  - Language and dependencies take time to master

Caveat: no login and terminal is stuck after init

- The kernel's *console* parameter is one thing, but your *init framework* is another. You may want to be explicit about the *allocated ttys* for login, etc.
- In some *init* framework it's almost trivial (once you know the problem) and you just add a line similar to another line "that works" in the file to the configuration file which defines the runlevels / getty program etc.
  - for example, in *finit-sysv*, add to */etc/finit.d/available/getty.conf* something like `tty [12345] /dev/ttyS0 linux noclear nowait` if ttyS0 is your desired serial.
- In systemd it can be somewhat less trivial, much more annoying (job timeout), and may require some more *systemd* experience, which is not always easy:
  - `[ TIME ] Timed out waiting for device ev-ttyS0.device - /dev/ttyS0.`
    `[DEPEND] Dependency failed for seri…rvice - Serial Getty on ttyS0.`

Caveat no login and terminal is stuck (systemd)

- As a sanity check, you may run (e.g. if you are running with QEMU) your system with a monitor (display), and see if you get the login there.
- This would most likely mean you would get a login on *tty1-tty6*
  - This can change.
  - On graphical systems, the graphical user interface will take over the ttys
    - e.g. tty7 in "legacy" systems (still common)
    - e.g. tty1 and tty2 in more recent systems
- If a *tty* works, you may modify */lib/systemd/system/getty-static.service*, and add your serial console interface to it, e.g.:

```
ExecStart=systemctl --no-block start getty@tty2.service
getty@tty3.service gettyy@tty4.service getty@tty5.service
getty@tty6.service getty@ttyS0.service
```

Caveat: more systemd getty adjustments

- To get the system to a *running* state (which is perfectly OK if you don't get it, some times, e.g. as in the serial-getty condition), upon caveats as mentioned earlier you would likely want to also disable the *serial-getty* service (or properly set *udev* and the relevant systemd parameters)
  # systemctl disable serial-getty@ttyAMA0.service
- If you run a headless system, or want to just allow login over the serial, you would want to avoid the spawning of tty1-tty6 (or so) in the *getty-static.service* file. This can (usually very slightly) affect boot time optimization

Caveat: multiple init frameworks and commands

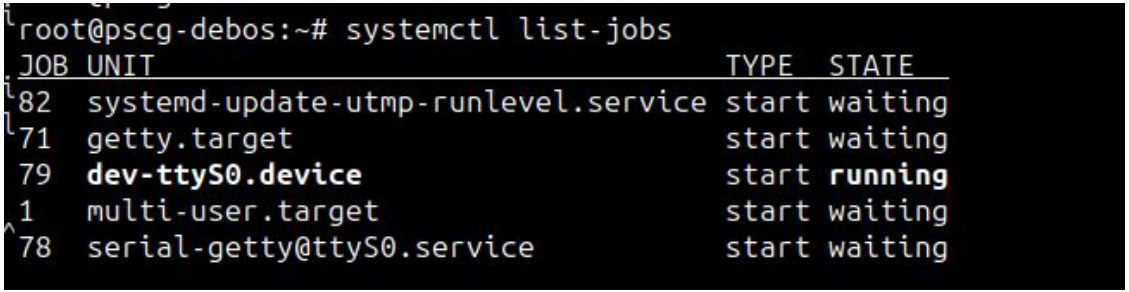
- Say you decided to install *systemd* and *finit* alongside.
- Some **super common commands**, such as *shutdown* and *reboot* are usually implemented as a symbolic link (@see *update-alternatives*) to the binary of the relevant framework
  - Which would then parse argv[0]...
- So, for example if you want to call **# shutdown**
  - and you first installed *finit* and then *systemd*
  - and you are running *systemd*
  - it is very likely that you will get an error message specifying that the init framework it is expecting is not up:

    `reboot: Failed connecting to finit: No such file or directory`
- There are various solutions to it, such as selecting one init framework (preferred!), calling *update-alternative*, and setting up the links yourselves.

Caveat continued

Even if you think all is good, you may still see that some services are trying to be run. This affects **boot time.** You can:

- **mask** the respective *serial-getty@tty<yourserial>* and see that the system will both go faster and go to a green state
- Install *udev* - systemd actually expects it for most of its auto-discovery shenanigans.



More troubleshooting

If things don't work, and you are not sure if it is your fault, *systemd*'s fault, or something else:

- if you see the *kernel logs* - you can already be quite happy.
- You can then set the kernel *cmdline* to include *systemd.unit=rescue.target*.
- Then if you get an interactive console, and can interact with it - you can make systemd work!
  - The recovery shell takes ownership of the stdin/stdout/stderr with "*tty-force*" on its su login shell.
  - You can look at the systemd source code for more details.

This kind of issues can be extremely time consuming! How valuable it is to use someone else's experience…

Takeaways: multiple init frameworks

- It's OK to play, but when you ship to your customers (and yes, I am referring to other engineers and developers you are supporting mostly), know your stuff!
- Decide on the init framework of your choice and plan accordingly
  - [and set up your services correctly depending on the framework of course…]
- Avoid mixtures and being too smart (or too lazy) for your own good
- *systemd* is powerful. But if you want to be effective in boot times - you need to understand very well how to control it.
- Other init frameworks are also powerful and you also need to understand them
- If you don't get login console, your console clears or doesn't clear etc. - don't panic. It's solvable.

Important observation about apt cache sizes

- Apt caches may take a lot of space
- So you may want to select your distros and components carefully
- You may also want to consider the tradeoff between:
  - Clearing the caches (and then requiring a big download upon an apt update)
  - Having big blobs of caches
- This may also lead to a consideration of *mirroring* an apt repository
- This may also lead to a consideration of storing the apt caches, and copying them (instead of an apt update) upon building
- Things can get quite complex, the moment you don't rely on "infinite storage and bandwidth"
- Tip: If you need something from a "heavy" repository/component, prepare it when building, and then disable the repo/component for future apt updates.

Reinstalling packages after cache modification

- Assume you have a running system and only remove the apt caches
- And then want to reinstall packages, after modifying some files
- The system would not be able to tell that a modification is indeed the same modification - and ask you to resolve changes.
- You want to know the strategy ahead of time - as working with the interface there is not very easy, even for experts. During *debootstrap* opening a shell is not a possibility (it will say that *x-terminal-emulator* is not found).
- So you need to be aware of the different *dpkg* flags, to set policies and install automatically (can do it in *etc* and can do it in each command)

```
A new version (/tmp/tmp.XoLn4rSeCx) of configuration file /etc/ssh/sshd_config is available, but the version installed currently has been locally modified.

  1. install the package maintainer's version    4. show a side-by-side difference between the versions  7. start a new shell to examine the situation
  2. keep the local version currently installed  5. show a 3-way difference between available versions
  3. show the differences between the versions   6. do a 3-way merge between available versions
What do you want to do about modified configuration file sshd_config?
```

Offline / reproducible builds

Recipe:

- Fetch sources ahead of time (kernel, busybox for initramfs)
- For a Debian based rootfs, *debootstrap* and save the resulting folder
- Chroot to the rootfs and *apt install*
  - And save the folder aside
  - Or - chroot to the rootfs and apt install - with *--download-only* and forcing flags save the caches aside
    - On the host, for further builds
    - Or on the target, for other reasons (next slide)
- There are multiple ways of achieving this, and the nice thing is being able to use the package manager capabilities, without networking

Cache reusing strategy on the target

- Saving the caches aside for target use can be extremely useful:
  - Assume you have a working root partition that you want to use
  - And want to keep it small, and read only
  - And you want to support A/B updates
  - And perhaps even using a recovery golden image that is small enough, but can leverage powerful capabilities (such as connecting to a server or to a user's or an operator's device), and you want to use it for getting a recent image after your current image failed terribly
- You can reuse caches in several different partitions
- And you can install more packages/remove more packages, etc. by simply modifying a shared cache
- And as a bonus - you will not be miserable fighting *systemd* links

Forwarding audio and video

- There are multiple display servers to select from
- And quite a few audio servers as well
- How you handle things are your choice
- The classics:
  - X11 (export *DISPLAY* environment variable)
  - PulseAudio (export *PULSE_SERVER* environment variable)
- The new ones:
  - Wayland/Weston
  - Jack, Pipewire
- Challenges with "the new ones"
  - Host configuration tends to be unstable and non-working. Requires some work
  - If you ask yourself for the meaning, try to use on your desktop, e.g. a modern Ubuntu version with almost every video conference product. You may prefer to revert to *X* on your *GDM…*

Forwarding audio and video use case

- This is beyond the scope of the currently allocated time slot
- However, a good working example of the rationale for the basics (or legacy basics) is available in my Linux on Macbooks - in and out of MacOS (Embedded Israel meeting #9) video

Security

Sorry folks, The PSCG does not treat security lightly! We either go obsessively rigorous about it, or we ditch it altogether.
In other words, security is discussed in other another module or another course

Where to take things next?

- Wherever you want, you are building your own products!
- We just gave some common tips, and illustrated the usage of *Debian* systems
- The rest is up to you.
- Build systems and enjoy them!
  - Or at least, struggle as less as possible

Perfect rootfs. OK. What do we do next (teaser)

- Handling read only file systems (requirements, robustness)
  - Partitioning
  - Overlaying
- Package update strategy
  - *Size* vs *Time* vs. *Network*
- Recovery/"Golden Image"
- OTA updates
- Handling the lower levels that boot them…
  - And then secure boot them
  - And then OTA them
  - And then add more features everywhere
  - And then do everything all over again, this is Embedded Linux!

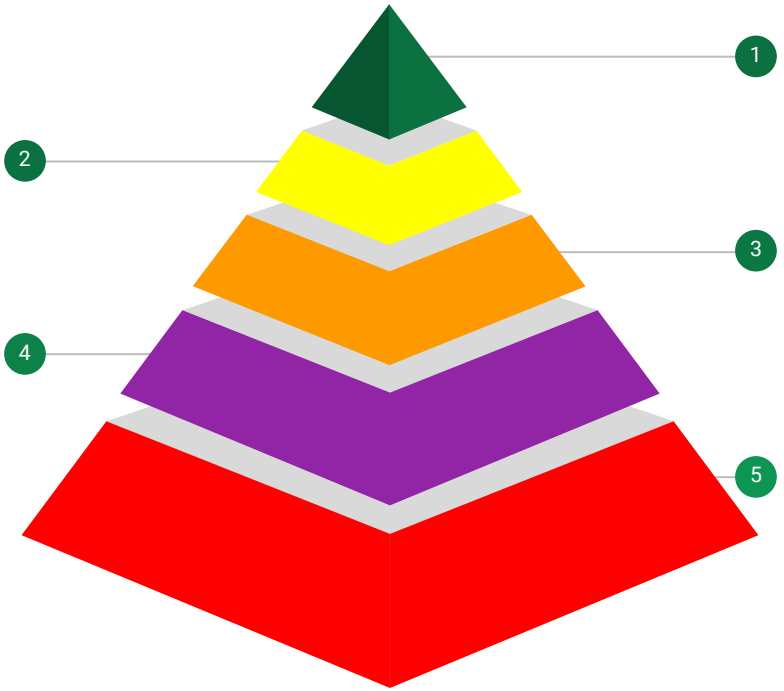
Learning Objectives - System Components

Provide the mechanisms of running binaries

Have a file system and set of tools that enable the user to run that application, including the components (e.g. libraries) that support it

Bootloaders to boot the kernel and initial file system

Bridge between the hardware factory boot code, and the prospective Operating System



Build and Run Applications/Binaries

An executable you can run, and (almost always) has dependencies.

GPOS (Linux) Kernel and Initial root filesystem

Provide the mechanisms to interact with the hardware, provide services to the mechanisms above, and implement multi-process and multi-user mechanisms

Hardware

This is what you can hold in your hands, or otherwise what "physically exists"

Summary

In this module we

- Explored *Debootstrap* (and cross-debootstrap)
- Explored various *Debian* based systems
- Learned about the *Debian* package management
- Understood the different package repositories and cached files
- Recommended some useful packages and configurations to start from
- Listed considerations in booting a debootstrapped system an *init* selection
- Discussed Sizing and performance constraints: Build-time and runtime
- Discussed some more common tricks and troubleshooting concepts
- [Extra: display and audio forwarding] // subject to host configuration

Continue the learning

- youtube.com/@ronmunitz - there are super relevant videos in there, and there will be more, as I will publish *PscgBuildOS*
- Follow me on LinkedIn
- Github: I have been using other platforms, but sometimes I used *ronubo* and some times *ronpscg* for some of the public talks stuff.
- If you want me to let you know when I publish the code, feel free to reach out to me after the talk, follow my posts in LinkedIn (I don't spam) or send me an email to ron@thepscg.com with the subject line [PscgBuildOS EOSS24] and I will let you know
- You are of course welcome to attend my training, but the entire idea in this "track" is to cover things that are complimentary to other training (mine and others')

PSCG

Premium Software Consulting Group

Thank You

For further training and consulting:

ron@thepscg.com

+972-54-5529466

/* backup slides */

PSCG

Introducing the PscgBuildOS

An educational (and operational) build system

Why another build system?

- Sorry to disappoint, but there is no extraordinarily good reason, if you are comfortable using the Yocto Project, you probably should.
- When I teach an Embedded Linux or Yocto Project course I like to demonstrate things before I give labs, and this makes for a great demo
- I found myself doing the same work again and again and again at different customers, so I figured out there is a demand for Debian builders
- I wanted to make a point that every Embedded Engineer can be a great Embedded Linux engineer and wanted to simplify the required tools, and "force" people to learn the very basics
  - Which is why everything is written only in *bash*
- I thought it would take me a couple of days to prepare.
  - *shrug*

Why another build system

- OK there are some good reasons
- You will build much, much, much, MUCH faster the things it was designed to build, using much less disk space
- And it will allow me to further give more recorded explanations, in a more spontaneous way, given that I am the one who wrote it, and I sort of know how I think
- And this allows me to do positive things for the community while enjoying it

Why another build system - a community story

- I promised in a meetup group that I am organizing that I will explain the rationale behind secure boot after looking at some *NVidia* Jetson boot flow
- And so I organized an OTA meetup which a well known company suggested to host.
  - But then they complained that a competitor was also invited to speak
  - And then they bailed out, perhaps because the POC who connected me left that company
- So it didn't happen, and I came back to the US and there were no more meetups that year
  - However, there was an Embedded Linux course for Embedded (Firmware) engineers…
  - And they made me understand, for like the millionth time that some very smart people have a Linux phobia. (Hmm… Startup validation!)
- Long story short - I decided to get to the OTA talk after covering pretty much everything else. I promised, and I do what I say.

Build system features

- Root filesystem builder
  - Debian/Ubuntu
  - Busybox based OS
  - Alpine (because someone asked a question about it in a meetup)
- Super easy to copy an example or just modify
- Supports multiple architectures
- Capable *initramfs* which also implements
  - an installer/flasher feature
  - OTA update mechanism
  - recovery/golden image mechanism
- Linux Kernel Builder
- Can be used to build other components and can build other Operating Systems

Open Sourcing the project and community work

- I was expecting to put it in Github already, but it is a bit too hacky
  - although you can see my explanations about it in the youtube channel I sometimes upload contents to
  - Naturally I always want to add "one more thing" and then…
  - If there is interest I will upload it as is
  - And if it helps people, I will also rename it ;-)
- I intend to continue this talk in the European *EOSS* - focusing on OTA/flashers/installers given the (simple) reference implemented in this distro.
- I am happy to add some more features, or have others collaborate on it
  - And I would be super happy to find the time…

Let’s see some of it in action

PSCG

Premium Software Consulting Group

Thank You

For further training and consulting:

ron@thepscg.com

+972-54-5529466